A Paradigm Shift! The Internet, the Web, Browsers, Java, and the Future of Computer Science Education[1]

Christopher M. Boroni, Frances W. Goosey, Michael T. Grinder, Rockford J. Ross
CS Department, Montana State University, Bozeman, MT 59717
ross@cs.montana.edu

Introduction — Get Ready to Shift!

Ready or not, here it comes! A paradigm shift with profound implications for computer science education is underway. The shift is away from a relatively static, localized paradigm of teaching and learning towards an interactive, dynamic, and non-localized paradigm.

The new paradigm is not totally unfamiliar. Various institutions have for some time been exploring educational activities based on interactive software, sometimes coupled with two-way interactive video for distance learning. These efforts, though, have largely been based on technologies that were not universally available in educational settings (e.g., Sun workstations) and sometimes expensive to acquire, thus limiting their widespread acceptance by others. Furthermore, even when the technology was fairly common (e.g., PCs with Windows 95) we have noted from long experience that the mere burden added onto the backs of busy instructors of downloading, installing (which never seems to go without some hitch), and learning one more new system has hindered the widespread adoption of many otherwise very fine educational software packages. But all of this has now changed, and rather abruptly at that. Downloading and installing software is no longer much of an issue, and even the local computer platform is of little importance. In short, due to the Web and its related technologies it is now possible for virtually any institution, large or small, to embrace the new paradigm with minimal effort and cost. Indeed, given the momentum of the Web, the paradigm shift[2] may be thrust upon us before we are ready to engage the clutch!

Numerous technologies are converging to effect this paradigm shift:

- the Internet, with its global computer interconnections;
- the World Wide Web, providing easy access to information virtually anywhere;
- browsers, such as Netscape and Internet Explorer, that allow nearly transparent access to the Web;
- HTML, which makes the development of universal, multimedia "hypertextbooks" and other documents possible;
- Java and the Java Virtual Machine, which provide an environment for the development of platform independent, interactive educational software that can be delivered via the Web through standard browsers;
- powerful notebook computers that can easily be carried to a classroom and connected to the Internet (or, if an Internet connection is not available, a hard disk or CD image of the relevant information) for use in a lecture;
- small, eminently portable computer projection systems that, too, can be transported to arbitrary classrooms and connected to a computer for display of computer output, allowing nearly any classroom to be adapted to the new paradigm; and
- improved interactive two-way video systems, providing for distance learning and remote student participation from properly equipped lecture halls.

Without question, the most influential of these technologies on the paradigm shift are the Web and the Java virtual machine. The elimination of platform dependence as a hurdle to producing good educational software systems that can be widely used without hassle is indeed a major achievement of profound import. We thus refer to the new paradigm as the *Web paradigm*.

There are many different facets to the paradigm shift that could be discussed, some possibly controversial. In the confines of this paper, however, we limit ourselves to the discussion of some of the implications of the Web paradigm on computing education. We should clearly state that our aim is not to discuss future computer science *curricula*. Instead,

[1] Support for some of the work described here has come in part from the National Science Foundation, grant numbers DUE-9354655, DUE-9552411, and DUE-9455588; and a 1997–98 MONTS grant from MSU.

[2] We computing professionals have been called lots of names, but one we haven't heard yet is *shiftless*!

our focus is on the teaching and learning *environment* that will result from the Web paradigm and—by implication—the profound influence it will have on any new curriculum models. In the Web paradigm, instructors will tend away from a traditional lecture style towards the role of a facilitator, and students will become more active in exploratory learning.

Any discussion of the Web paradigm that did not include some exciting, dynamic examples would certainly not be very satisfying. Thus, the accompanying talk will be liberally sprinkled with projected demonstrations of the major concepts using relatively inexpensive technology available today (and sure to be more accessible and affordable tomorrow). So, unfasten your mindbelts and enjoy the ride!

The Common Teaching Paradigm — A Static World

Surely the one professor in the author list for this paper is not alone in having honed his lecture techniques over the years[3], accumulating some high teaching evaluations and the occasional "teacher of the year" award in the process, only to be forced to admit that attendant increases in student understanding and retention have not been nearly so evident. The problem, at least in our view, is that current, generally static, lecture-based teaching paradigms seem to have reached the limit of their effectiveness. The most dramatic increases in (apparent) teaching effectiveness have come through the development of a dynamic lecture style, with lots of interactive illustrations on the blackboard (e.g., program walkthroughs, demonstrations of operations on ADTs, and the like), and efforts to engage the students in ongoing dialog. The students love it. The problem is, they leave the dynamic lecture environment at the end of the period, often with dire consequences (this situation has been noted in the literature—see for example [9]:

- Once the lecture is over, students return to the static world of textbooks and lecture notes as their primary—and perhaps only—sources of information.

- Notes dutifully taken in the lecture are often more than a little confusing. . .how does one capture the dynamics of an animated lecture in notes?

- Even during lecture, the dynamic approach is prone to errors. If a student asks a question, or a mistake is caught, backing up to a particular point to restart is often difficult or impossible.

- It is well known that repetition is one of the keys to learning. Yet, there is no time in the standard lecture paradigm for much repetition. This is usually left to the students in the form of exercises, which is where many students have difficulty, because it is precisely the dynamics and animation of the lecture that they need to have repeated[4].

[3] At least *he* has the illusion that his teaching has improved substantially!

The result is that students who thought they understood perfectly well during lecture often wind up confused once they are on their own again. How much better it would be if the animations of the lecture could be performed by a student as often as necessary in an interactive, reversible, repeatable, exploratory, and error-free environment. This is the promise of the Web paradigm (born out by individual animation projects at MSU and elsewhere).

The Web Paradigm — A Student's Scenario

In contrast to the situation presented in the previous section, consider this scenario. A student is learning about operating systems. As a supplement to standard educational materials, such as a textbook, the student has access to a computer with a Web browser (e.g., Netscape). Connecting to Web-based course materials through the browser, the student follows a link to the current topic in operating systems. While reading about the instruction fetch and execute cycle of a computer, the student finds a special *concept animation* link; following this link invokes a Java applet—with which the student can interact, changing various parameters and input values—that effectively demonstrates the instruction fetch and execute cycle of a computer in the presence of interrupts, showing how a process is preempted, shunted to a priority queue, and replaced with another running process by the operating system.

While experimenting with the operating system applet, the student may realize that the priority queue abstract data type (ADT) concept is still unclear. Following an associated link in the browser leads to a description of this ADT along with special program and algorithm animation links. Clicking on the *algorithm animation* link retrieves an animation applet showing a priority queue both in binary tree form and in array form. Again, the applet provides the capability for interaction, allowing the student to exercise the ADT operations (e.g., insert and delete) as the ADT is learned.

At some point, the student may well desire to view the actual program code implementing the ADT, in which case a *program animation* applet can be invoked. This applet, too, provides interaction possibilities to the student. The student can execute the program a step at a time, watching variable values being changed, recursive calls being made, parameters being passed, and so forth. At any puzzling juncture, the animator allows the student to reverse execution arbitrarily far to repeat a sequence of steps. Execution cost is displayed as well, providing a mechanism for determining the time complexity of the underlying algorithm. If the student desires to alter the program to see what happens, or indeed to enter an entirely new program for animation, an editor and a compiler in the applet can be invoked for this purpose.

Exercises (e.g., to test the student's understanding of the priority queue ADT) and experiments (e.g., to determine the

[4] How many walkthroughs of various concepts have we done during office hours for the lost student?

time complexity of the insert operation in the priority queue ADT) are interspersed in the document, and may even be assigned for the student to do. At the end of a few sessions, the student leaves with a clear, intuitive picture of how the hardware of a computer (in this case the instruction fetch and execute cycle with interrupts) interacts with the software (i.e., the operating system) to allow multiple users to execute concurrently on a single processor. The student also achieves a better understanding of the priority queue ADT, the implementation of this ADT as a program, and the use of this ADT for managing concurrent tasks in an operating system. Further, the student is beginning to see the integration of all of these individual concepts.

And now consider this: the student of this scenario does not need to be in a formal classroom. Indeed, the student may not even be at the university. This *could* be a regularly enrolled student working, say, in a supervised laboratory. But the student could also be working in a dorm room, learning at home far from the university, working in a special setting because of a disability, studying with no formal ties to a university, or learning in a special high school class for which there is no local teacher.

Is this paradigm far-fetched? Not in the least! Examples of most of these animations have already been constructed and demonstrated by the the authors and other researchers in computer science animation. What is new is the universal delivery mechanism presented by the World Wide Web, Java, HTML, and common Web browsers that allows this work to be integrated into a single system and made available to nearly anyone anywhere[5].

The Scenario Again — An Instructor's Viewpoint

Of course, the capabilities afforded the student in the above scenario are also available to the instructor; indeed, the instructor may have had a hand in developing some of the animations used by the student (or, the instructor may simply be pointing the student to work already done and available on the Web). The instructor may be fortunate enough to have a classroom with a computer, projection device, and internet connection for holding lectures[6]. If not, he or she can pack a notebook computer and a portable projector to a classroom. In supplementing the lecture, the instructor can access the same concept, algorithm, and program animations while engaging the students in a dialog about the instruction fetch and execute cycle of a computer that the students will later study in more detail on their own time. Instead of lectures based on static textual material, the instructor is able to facilitate a highly visual, motivating, and interactive presentation of the material that is accurate, repeatable, and that lends itself well to questions and active learning. Remember, students have access to the same Web-based materials that the instructor has been using, freeing them from the distraction of taking copious notes and allowing them to concentrate instead on the dynamic classroom activities, activities that they can repeat later on their own time and which they could not capture in notes in any case.

The New Paradigm — Beginning the Shift

Where do we stand with respect to the paradigm shift? Nearly every computing curriculum is using the Web to some degree in teaching and learning. While Web use may be limited in many cases to relatively static activities, such as the dissemination of class syllabi, distribution of mail, posting to news groups, and occasional assignments that require research using the Web, this initial phase is quite important in that it acquaints instructors and students alike with the environment of the Web. As the paradigm fully engages, its interface will thus be entirely familiar and natural to use.

Examples can be found at www.cs.montana.edu/cs/-courses. Computer science students at MSU are required to construct a personal Web page in their first computer science course. Each course also has a Web page for the course syllabus, which in some cases is quite detailed and may include individual lectures or at least summaries of class activities. Try, for example, www.cs.montana.edu/cs/courses/418f97. In addition to being required to access the class Web page for syllabus information, all computer science students at MSU are required to read their e-mail daily (in case an instructor needs to contact an individual student), and to read a special class news group daily (in case an instructor needs to convey information to the entire class).

These uses of the Internet and the Web at MSU are likely representative of the types activities common in most computing departments. Again, it is important to note that one result of these activities is that students and instructors alike become very comfortable with the Web and its interface through common browsers.

Concurrent with the rise in the incorporation of these relatively simple kinds of Web-related activities into the curriculum have come advances in classroom technology that allow instructors to demonstrate computing concepts during lecture. Portable projection tablets that allow computer output to be displayed on a screen using a standard overhead projector have been around for some time. Now, notebook computers and bright, portable, high resolution projectors make classroom presentations even more compelling. There is also a general move on campuses to construct new, "high-tech" classrooms that are self-contained in this respect. At MSU the Computer Science Department has just moved into the new Engineering and Physical Sciences Building that was designed from its inception to be "computer ready." One large lecture hall and a number of smaller classrooms are equipped with very high resolution rear projection systems, a computer, sound, and the usual array of VCRs, laser disk players, and other toys. The rooms are also equipped with cameras, monitors, and student microphones for distance

[5] Finally!

[6] Retrofitting classrooms with a network connection is usually not a difficult task, and should be done as part of efforts to network an institution; it goes without saying that all new classrooms should be constructed with network connections built in.

learning. All have network connections. Thus it is possible to give Web-related lectures and engage the class in Web activities. With such an environment, instructors have quickly taken to modifying their lecture styles to be Web-based, including HTML “slides” of lectures, demonstrations of concepts reached by following active Web links, and displays of specialized software packages[7].

Will there be resistance to shifting to the Web paradigm? We don’t think so. When Web technology is readily available in the classroom instructors seem eager to incorporate pedagogical innovations. One supposes that this comes from a long frustration with trying to describe concepts that could be explained so much more easily if only they could be directly demonstrated, as they can be with network connections. In short, our experience indicates that instructors are already contemplating new styles of presentation just because they now can through use of the Web directly in the classroom. This bodes well for the paradigm shift.

The New Paradigm — The Middle of the Shift

The current use of Web technology in computing education, as outlined in the previous section, scarcely begins to tap the possibilities. Student oriented Web activities are still largely static in the sense that the students deal mostly with fixed HTML documents and have few opportunities to learn about topics through hands-on, interactive, and modifiable modules. The advent of Java and the Java virtual machine has changed this forever. It is now possible to develop interactive animations of fundamental computing concepts and incorporate them into Web pages that instructors can use for lecture demonstrations and with which students can interact in a myriad of ways to enhance their learning.

The research community actually has a jump on creating such animation modules. Work has been underway for a number of years by various researchers in computer visualization of computing concepts. As noted earlier, some extant systems are quite good, but they all suffer from the plague of platform dependence and the requirement that a user download, install, and learn the system. All of the active researchers in this area have now refocused on the Web and applets as the media of choice for development of animation software. In the following, we explore the three main types of animation: program animation, algorithm animation, and general concept animation.

Program Animation refers to the visual, interactive presentation of a program on a computer monitor for study and experimentation. At first glance, a program animator appears similar to a program debugger. In practice, however, a program animator is quite different than a debugger, in that it is oriented towards *learning* about programs and algorithms rather than *debugging* programs. Over the past number of years, our research group has developed a robust and well-developed program animator. The animator allows a student to choose a Pascal program from a library of programs listed in a **File** pulldown menu (e.g., the entire set of programs in a textbook) for animation and then step through the program with complete control as nearly all aspects of program execution are displayed in scrollable panes of the main animation window. Stepping options include step with pause (which allows a student to study the effect of executing a particular statement and pondering what the next statement to be executed should be before highlighting moves on to that next statement), step without pause, and continuous execution, among other options. Particularly noteworthy are features for measuring run time costs (for time complexity experiments) and for reversing execution an arbitrary number of steps to allow repeated study of a particularly puzzling portion of the program, features that are important to both the instructor using the animator to supplement a lecture and students reviewing concepts on their own.

The animator comes with an editor and a compiler that can be used by instructors to develop a library of programs for students to study and by students to modify existing programs or to develop entirely new ones. The animator depends on a unique virtual machine (in the same vein as the Java virtual machine) that is capable of executing programs in its machine language both in the normal (forward) direction or in reverse. By virtue of this virtual machine—called the E-machine—the same animator can be used to animate programs of any other language as well, once a compiler has been written for that language that targets the E-machine. Subsets of C and Ada have been implemented, and an extensive Java compiler is in the works, the PhD thesis of the first author.

The complete animator currently runs on Windows 95 and NT platforms, and thus suffers from the platform dependence and downloading issues mentioned earlier. Nonetheless, the system has been used with success at other institutions, including Duke University, and has been downloaded by numerous people around the world. There is thus little doubt that the program animator (and other projects like it) will find widespread use in the new teaching and learning paradigm once it is completely ported to Java and the Web. This is one current focus of our research group. A snapshot of an animation of a recursive program to compute factorials is given in figure 1. The animator has been developed through a series of MS projects and NSF grants (see footnote 1).

Algorithm Animation refers to the visual, interactive, graphical representation of an algorithm on a computer monitor. For example, there are many animations of sorting algorithms that start with a display of bars of various heights in random order (representing a series of random numbers) and then highlight the bars changing positions according to the method of the particular sorting algorithm until the bars are all rearranged from shortest to tallest. Many more so-

[7] Remember that with a notebook computer, a computer-output projector, and an internet connection the same effects can be realized, albeit with some added effort.

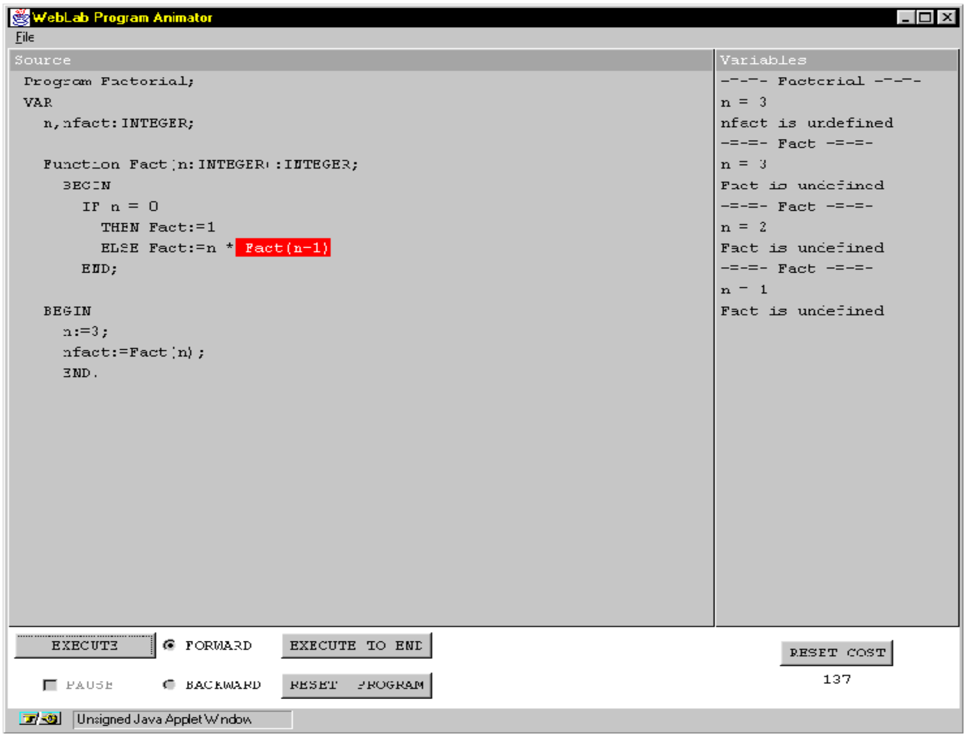

Figure 1: Snapshot of a Program Animation of a Recursive Factorial Program

phisticated animations of other algorithms have also been done, and many are available as Java applets on the Web already. These range in capabilities from animations that are self-contained and simply show an algorithm in action, to animations that are truly interactive, allowing students to exercise various operations (e.g., insert a new element), to advanced animation systems that actually allow the user to create altogether new animations of arbitrary algorithms. Some animations allow both a code view (so that one can observe the program underlying the algorithm) as well as a graphical view of the algorithm.

Projects of note by researchers currently active in algorithm animation research include Balsa and Zeus by Marc Brown [4], GAIGS by Tom Naps [8], and X-Tango and Polka by John Stasko [6, 10]. Each of these researchers has been active in algorithm animation for a number of years, and their earlier systems ran (as was usual) on only specific platforms. All, however, have turned their attention to the Web and applets as a way to surmount the problem of platform dependence. Both program animation applets and algorithm animation applets figure to play an exciting role in Web-based teaching and learning environments.

Many others—too many to mention here—have tried their hand at algorithm animation. A snapshot demonstrating a binary search tree algorithm animation, borrowed from the very interesting work of Doug Ierardi and Tan-Wei Li of the University of Southern California, appears in figure 2. Their animations can be found at

http://langevin.usc.edu/BST/

Links to other interesting sites include

- www.cc.gatech.edu/gvu/edtech/ (John Stasko's work);
- http://gaigs.cmsc.lawrence.edu (Tom Naps's work);
- www.research.digital.com/src/publications (search on Marc Brown's name);
- www.research.digital.com/src/staff/najork/bib.html (Marc Najork's work);
- www.cs.brown.edu/cgc/cgc-brown.html (the Mocha project [1]);
- www.gamelan.com (follow the links to the computer science education section).

Concept Animation, for lack of a better term, refers to all other types of animations of computer science topics. These include animations that illustrate the parsing process in an LL or LR parse, the visualization of the instruction fetch and execute cycle of a computer, the presentation of nondeterministic automata, and so forth. Such animations tend to be quite labor-intensive to develop. John Stasko has done much pioneering work through his Tango and Polka systems (see the link to his site given earlier). The work of Susan Rodger

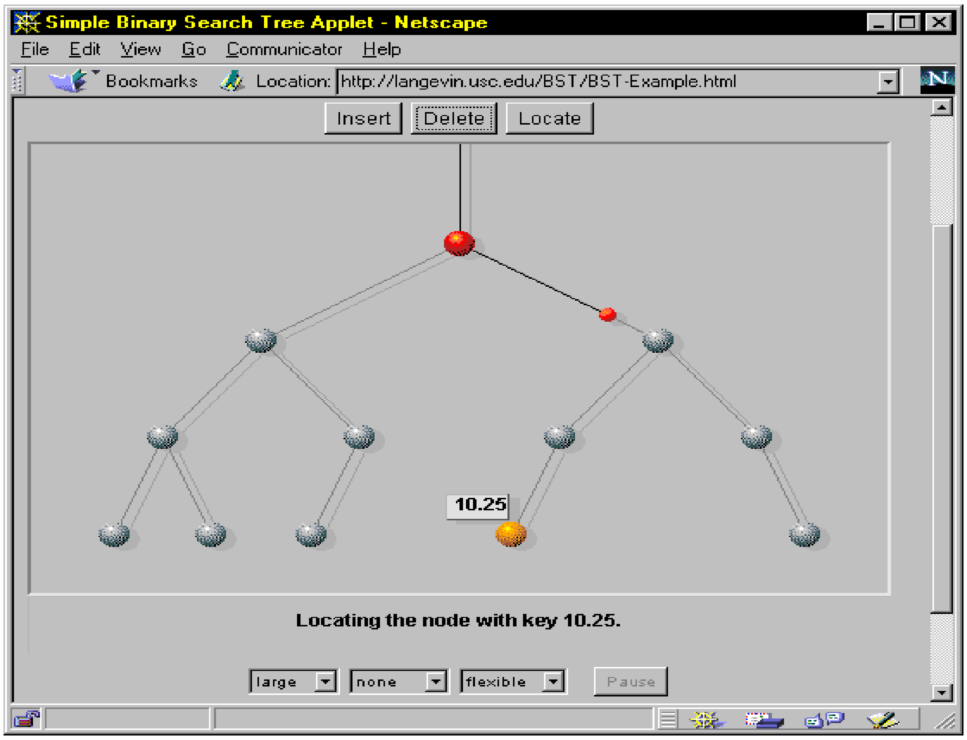

Figure 2: Snapshot of a Binary Search Tree Animation Applet

[3, 2] and Alan Biermann [7] are noteworthy in this regard as well. Links to their work can be found, respectively, at

- www.cs.duke.edu/∼rodger/
- and www.cs.duke.edu/∼awb/

Susan Rodger has been particularly active in porting her earlier work to the Web.

Again, others have developed animations and more appear regularly on the Web. Other suggested sites include

- http://info.latech.edu/∼watson (the Watson Project) ;
- http://sol.pace.edu/webvis/ ;
- http://trp.research.apple.com/EdEconomy/ ;
- www-csli.stanford.edu/hp/ .

In our own research group, the third author has as his PhD thesis topic the animation of the theory of computing. The goal is to provide interactive animations of the fundamental concepts of the theory, including problem reductions and NP-completeness. As a pilot project, he has implemented, class tested, and published [9] a comprehensive Java finite state automaton animator applet that can handle both deterministic and nondeterministic automata. A snapshot of an animation of a finite state automaton for recognizing valid dollar amounts is given in figure 3. From within this flexible animator, instructors and students can select from a potentially large number of automata to animate, and they can modify existing automata as well as create entirely new ones.

Program animators, algorithm animators, and general concept animators cover the gamut of computer science topics. There will always be new animations to be done and old animations to improve. However, techniques are sure to be honed and repertories built up, making interactive, exploratory Web-based teaching and learning ever easier to incorporate into the curriculum.

The Future—Completing the Shift

So, what does the future look like with regard to the paradigm shift to Web-based, interactive teaching? While it is not entirely clear, the scenario given at the start of this paper presents one view, which is the objective of our current research efforts. Nearly all of the pieces are in the making or are already floating around the Web to realize the scenario. What is needed is a number of concerted efforts to coalesce the ideas embodied in program animation, algorithm animation and concept animation into dynamic, interactive, multimedia hypertextbooks and laboratory materials in the spirit of the scenario. Such a "book" may indeed be too unwieldy for transport across the Web. Fortunately, the same technology can be bundled onto a single CD (or in the near future, a digital video disk—DVD) and packaged with a more tradi-

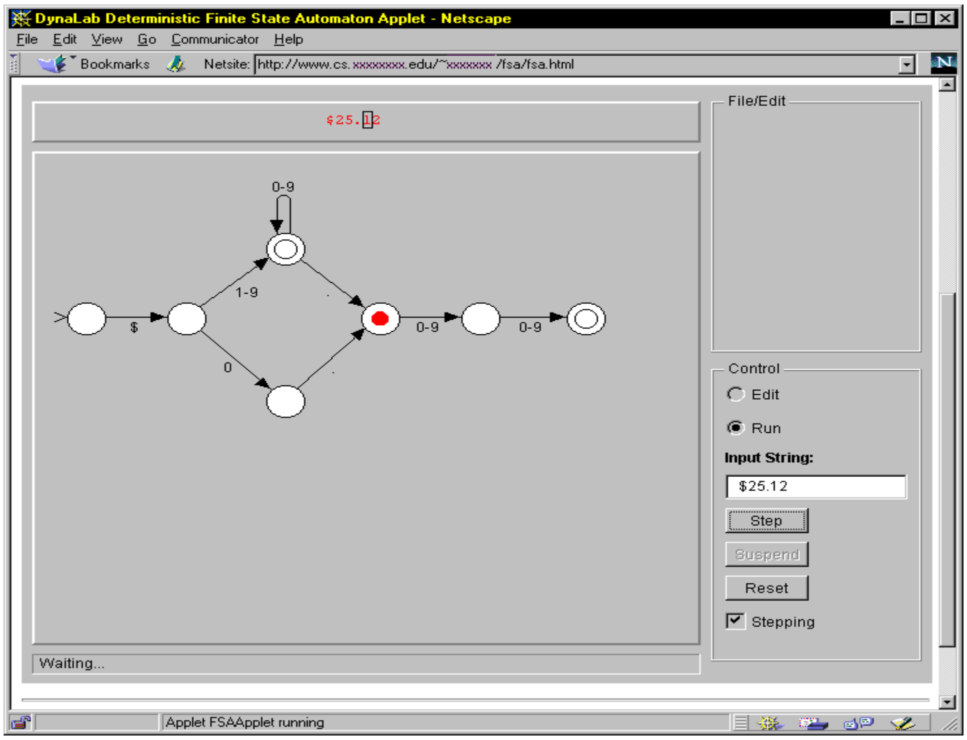

Figure 3: Snapshot of a Finite State Automaton Concept Animation Applet

tional textbook for little cost and accessed by standard Web browsers in the usual fashion. New editions can be released regularly, perhaps every semester, and the fixed CD material can be supplemented with links to other Web sites.

With such a system, instructors can teach and students can learn in the manner described in the scenario. One of the current focuses of our research group is to develop such a Web- and Java-based hypertextbook (including sound where helpful), the beginnings of which can be found at www.cs.montana.edu/~weblab/.[8]

With the Web and its emerging technologies comes an added richness to any endeavor of the kind described here. For example, works can be tailored with links to meet special needs, such as addressing the issue of the decreasing numbers of women in computing curricula (e.g., a link can be included to The Ada Project (TAP), www.cs.yale.edu/-HTML/YALE/CS/HyPlans/tap). And then there are the prospective changes to distance learning: with streaming audio and video coupled with cameras and microphones connected directly to a personal computer (all of which are presently available, if not quite ready for prime time) it will be possible for students to be involved in lectures from virtually any location—dorm rooms, high school classrooms in remote, rural locations[9], and so on, not just in specially equipped remote classrooms.

...and the list goes on. While it is hard to imagine everything that lies in store, we can be certain that teaching and learning will tend towards more interactive, Web-based methods. Our teaching styles, pedagogy, and methodologies will all be affected, we think for the better.

Hurdles

There are certainly some hurdles to be jumped during the paradigm shift. Not everything is quite in order. In every case, though, it appears that solutions are on the horizon. Following are some examples.

- HTML is not yet sophisticated enough for writing comprehensive computing textbooks. For example, greek symbols, formulas, and other technical notation are difficult to include. At present, such technical information is incorporated via gif files (i.e., as pictures) (there are some Java applets that aid the hypertextbook writing process as well). The solution should come as HTML standards are resolved and implemented. HTML3, for example, promises to go a long way towards addressing these problems.

[8] It should be noted that even this idea is not new. The *Algorithms* book by Cormen, Leiserson, and Rivest, for example, was reproduced with a similar, albeit more limited, vision on a CD ROM with hypertext links and algorithm animations. It works only on Macintosh computers and hence has seen far less use than it is worth, due again to the platform dependence plague.

[9] What do we mean by "rural?" The third author of this paper graduated in a high school class of *nine* in the remote town of Lima on the Montana-Idaho border.

- Java still has some teething problems. It is not yet standardized, and Java applets can still run quite slowly. Furthermore, the big issue of being able to perform standard file operations (e.g., saving a file created by an applet) has not yet been satisfactorily resolved for security reasons. Solutions to all of these problems are being tested by Sun, and are sure to come, based—if on nothing else—on the momentum of the big commercial players who are eager to be able to develop and market their products with complete platform independence and no ties to a particular company's whims (i.e., Microsoft).

- The Netscape/Microsoft/Sun wars continue. Platform independence is a fine concept as long as it does not dilute some powerful company's profits. How this situation will shake out is uncertain, but again, we cite the clear momentum behind Java and the Java Virtual Machine as an indication that browsers will continue to support this combination.

Thus, even though there are problems, there will also be solutions that ensure the viability of the Web paradigm.

Summary

So here we stand, on the shifting edge of a paradigm. Will the new Web paradigm be better than the current paradigms? That remains to be seen, but preliminary results from our own formal evaluations of the animations developed by our team [9] are very encouraging. Some evaluations by others are less conclusive [5]. In every case, though, one observation is consistent: students are turned on by interactive, animation-based learning. That alone to us is a strong endorsement. If we can attract, excite, motivate, and retain students—particularly those currently underrepresented—we will be tickled with the results.

Curriculum changes will occur, as they have regularly in the young history of computing education. Never before, though, has technology played such a key role in leading us to rethink our views of teaching and learning. The Web paradigm shift will have both subtle and profound effects on our pedagogy and our curricula. These are indeed exciting times!

References

[1] James E. Baker, Isabel F. Cruz, Luis D. Lejter, Giuseppe Liotta, and Roberto Tamassia. Mocha. Technical report, Brown University, www.cs.brown.edu/-cgc/cgc-brown.html/, 1996. This Web page describes the Mocha system.

[2] Anna O. Bilska, Kenneth H. Leider, Magdalena Procopiuc, Octavian Procopiuc, Susan H. Rodger, Jason R. Salemme, and Edwin Tsang. A Collection of Tools for Making Automata Theory and Formal Languages Come Alive. In *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 29, number 1, pages 15–19, March 1997.

[3] Stephen A. Blythe, Michael C. James, and Susan H. Rodger. LLparse and LRparse: Visual and Interactive Tools for Parsing. In *Twenty-fifth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 26, number 1, pages 208–212, March 1994.

[4] Marc H. Brown. The 1993 SRC Algorithm Animation Festival. Technical Report 126, DEC Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, July 1994. This tech report lists numerous references to other algorithm animation systems.

[5] Michael D. Byrne, Richard Catrambone, and John T. Stasko. Do Algorithm Animations Aid Learning? Technical Report GIT-GVU-96-18, Georgia Institute of Technology, Atlanta, GA 30332-0280, August 1996. The results here are not negative, but they are inconclusive.

[6] David Carlson, Mark Guzdial, Colleen Kehoe, Viren Shah, and John Stasko. WWW Interactive Learning Environments for Computer Science Education. In *Twenty-seventh SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 28, number 1, pages 290–294, March 1996.

[7] Christopher Connelly and Alan W. Biermann. Tools for Visual Exploration of Scope and Parameter Passing in a Programming Language Course. In *Twenty-seventh SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 28, number 1, pages 63–67, March 1996.

[8] Thomas L. Naps and Jeremy Stenglein. Tools for Visual Exploration of Scope and Parameter Passing in a Programming Language Course. In *Twenty-seventh SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 28, number 1, pages 305–309, March 1996.

[9] Rockford J. Ross. WebLab Papers. Technical report, MSU, http://www.cs.montana.edu/∼weblab, 1997. The WebLab project page provides Postscript copies of all relevant WebLab papers, which are too numerous to cite here.

[10] John T. Stasko. Using Student-Built Algorithm Animations as Learning Aids. In *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 29, number 1, pages 25–29, March 1997.